SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4831

Ostvarenje imperijalističkog optimizacijskog algoritma u programskom sustavu za evolucijsko računanje

Kristijan Jaklinović

Zagreb, lipanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 5. ožujka 2017.

ZAVRŠNI ZADATAK br. 4831

Pristupnik: **Kristijan Jaklinović (0036479316)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Ostvarenje imperijalističkog optimizacijskog algoritma u programskom sustavu za evolucijsko računanje**

Opis zadatka:

Opisati probleme kontinurane optimizacije i vrste stohastičkih optimizacijskih algoritama. Istražiti inačice optimizacijskih algoritama temeljene na imperijalističkom natjecanju te dostupne primjere uporabe algoritama u problemima iz inženjerske prakse. Ostvariti programski sustav za optimizaciju proizvoljnog problema korištenjem paradigme imperijalističkog natjecanja uz pomoć postojećeg programskog okvira za evolucijsko računanje. Ocijeniti učinkovitost ostvarenih algoritama s obzirom na vrstu problema i parametre postupka. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 9. lipnja 2017.

Mentor:
Izv. prof. dr. sc. Domagoj Jakobović

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

Hvala mentoru izv. prof. dr. sc. Domagoju Jakoboviću na strpljenju i pružanoj pomoći.

SADRŽAJ

1. Uvod

Problem čekanja javlja se svuda oko nas, bilo to studentski raspored koji ima prevelike pauze između predavanja ili produkcijski problem u industrijama. U moderno doba vrijeme je sve više dragocjeno, stoga poslovi zahtijevaju promišljen i smislen redoslijed izvođenja jer uvelike utječu na maksimizaciju profita gotovo bilo koje tvrtke. Ponekad, ovisno o problemu, nije ni moguće pronaći onaj redoslijed izvođenja poslova koji će predstavljati savršeno rješenje; stoga pokušavamo pronaći ono koje nam je zadovoljavajuće. Čovjek taj problem može razriješiti samostalno samo onoliko koliko mu to dozvoljavaju njegove sposobnosti. Ali kada je količina poslova prevelika, taj problem za čovjeka postaje nerješiv. Savršen primjer takvog problema bila bi industrijska proizvodnja čija kvaliteta proizvoda ovisi o vremenskom rasponu između dva proizvodna procesa. Broj strojeva je ograničen te je potrebno određeno vrijeme da jedan proizvodni proces završi, te vrijeme različitih procesa nije nužno jednako. Kako bi se maksimizirala količina proizvedenih jedinki i kvaliteta proizvoda potrebno je napraviti takav redoslijed izvođenja koji će upravo to omogućiti. Optimalno rješenje ima značajan utjecaj na niz čimbenika kao što je sama cijena proizvoda koja direktno utječe na ostvareni profit. Amplituda posljedica odražava se uvelike na tržištu, stoga uviđamo koliko je zapravo važno kvalitetno upravljati redoslijedom izvođenja poslova na resursima kojima raspolažemo. U ovom radu nastoji se objasniti optimizacijski problem te se predlaže algoritam za njegovo rješenje. U drugom poglavlju pokušava se objasniti problem optimizacije i pristup njegovom rješavanju. Opisana je složenost problema te način na koji razne metaheuristike generaliziraju taj problem. U trećem poglavlju opisan je svaki korak rada stohastičnog metaheurističnog algoritma koji rješava problem optimizacije, a u četvrtom poglavlju nalazi se pregled dobivenih rezultata tog algoritma. Peto poglavlje obuhvaća sva ostala sa kratkim osvrtom te zaključkom.

2. Problem optimizacije

Već je spomenuto gdje optimizacija sve pronalazi svoju primjernu i koliko je bitna. No put do optimalnog rješenja nije lagan, a ponekad, zapravo najčešće u primjeni, optimalno rješenje nije niti pronađeno. Prostor pretrage može biti toliko velik i kompleksan da pronalazak optimalnog rješenja zahtjeva nerazumno puno vremena. Stoga prihvaćamo sva dobra rješenja koji ne moraju nužno biti i optimalna, ali su relativno brzo pronađena. Pristup rješavanju takvih problema svodi se na sljedeće:

Slika 2.1: Općeniti proces donošenja odluka

Identificiranje problema od nas traži da pomno promotrimo problem koji rješavamo, da primijetimo koje ulazne vrijednosti dobivamo, kojim parametrima raspolažemo, koji su naši ciljevi rješenjem tog problema te koja ograničenja su prisutna na putu. Tijekom identifikacije teško je odvojiti koje točno informacije smatramo relevantnima, tj. koji bi sve parametri mogli utjecati na kvalitetu rješenja. Stoga ovaj korak nije nužno precizan.

Modeliranje problema je najbitniji korak jer kvaliteta rješenja uvelike ovisi o modelu koji će predstavljati naš problem. Modelom pokušavamo što više pojednostaviti stvarnost. Što naš model vjernije prikazuje problem na apstraktnoj razini to će kvaliteta rješenja biti bolja. Tu obično imamo dva pristupa kod modeliranja. Imamo pojednostavljeni model kojim pokušavamo što vise apstrahirati problemi imamo egzaktni model koji što vjernije prikazuje problem.

Prilikom optimizacije teško je odabrati najbolji pristup za rješenje pojedinog problema jer ne znamo koji će biti najbolji. Trenutno postoji jako puno pristupa rješavanju optimizacije kao što je kreiranje algoritma koji će rješavati neki specifičan problem ili

možemo koristiti već neki od postojećih algoritama za rješavanje sličnih problema.

2.1. Opis problema kontinuirane optimizacije

Optimizacijski problem definiran je sa s (S, f) gdje je S skup svih mogućih rješenja, a $f : S \to \mathbb{R}$ funkcija koju treba optimizirati gdje je S određen ulaznim parametrima. Pritom nam je glavni cilj rješavanja optimizacijskog problema pronalazak globalnog optimuma, tj. jednog od njih ukoliko ih funkcija ima više. Rješenje $s^* \in S$ je prihvatljivo, odnosno globalni optimum ako zadovoljava sljedeći uvjet:

$$\forall s \in S, f(s^*) \leqslant f(s), (\text{problem minimizacije}) \tag{2.1}$$

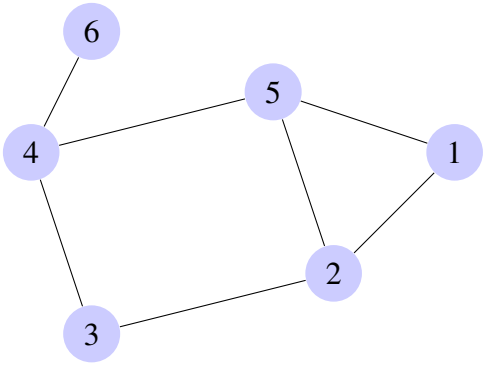
$$\forall s \in S, f(s^*) \geqslant f(s), (\text{problem maksimizacije}). \tag{2.2}$$

Brzina pronalaska rješenja je u korelaciji sa složenosti algoritma koji rješava taj problem. Prema složenosti, probleme dijelimo na lako obradive te teško obradive. Za lako obradive probleme postoje algoritmi koji taj problem rješavaju u polinomijalnom vremenu, dok za teško obradive nije poznat algoritam za rješavanje u polinomijalnom vremenu.

Svaki se optimizacijski problem dade svesti na problem odluke, tj. najjednostavnije "da/ne". Odluka "da" je donesena u slučaju kada funkcija cilja neke jedinka bude manja, tj. ide prema optimumu (problem minimizacije), inače je odluka ne za gore rješenje od postojećeg. S obzirom na složenost, probleme odluke dijelimo na razred P, NP, NP-kompletan, NP-težak.

2.2. Razred P

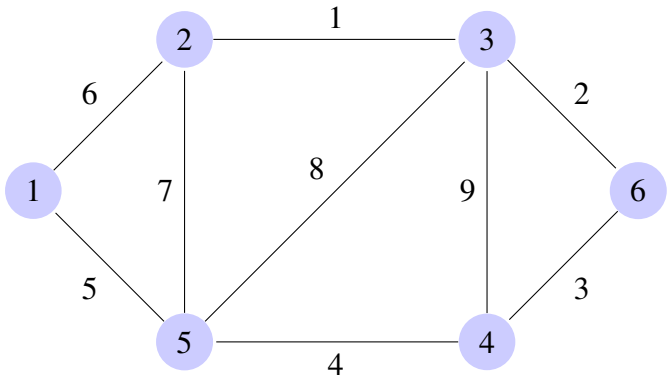
Razred P odnosi se na probleme odluke koje je moguće riješiti deterministčkim strojem u polinomijalnom vremenu, tj. složenost najgoreg slučaja je ograničena polinomijalnom funkcijom. Primjer P problema je npr. problem najkraćeg puta između dvije točke u grafu tako da zbroj težina na putu bude najmanji.



Slika 2.2: Problem najkraćeg puta

2.3. Razred NP

Razred NP predstavlja problem odluke koje je moguće riješiti nedeterminističkim strojem u polinomijalnom vremenu. Primjer takvog problema su svi P problemi, te npr. TSP (*Traveling Salesman Problem*), tj. problem trgovačkog putnika. TSP možemo vizualizirati kao graf čiji čvorovi predstavljanju gradove, a grane težinu, odnosno udaljenost između međusobno povezanih gradova. Kao rješenje problema tražimo onu rutu koja će nam omogućiti da u što kraćem putu obiđemo sve gradove.



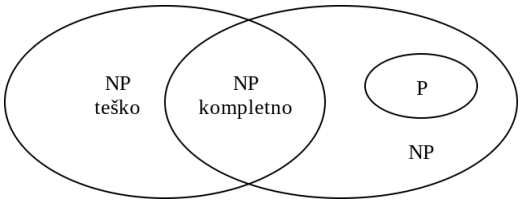
Slika 2.3: Problem trgovačkog putnika

2.4. Razred NP-kompletno

NP problem odluke X je NP-kompletan ako se svi ostali problemi u NP daju svesti na X u polinomijalnom vremenu. Za slučaj da je pronađen algoritam s polinomijalnim vremenom za NP-kompletan problem, on može riješiti sve probleme u NP. Tada vrijedi $P = NP$.

2.5. NP-teško

Problem je NP-težak ako algoritam za njegovo rješenje može biti preveden u algoritam za rješavanje NP problema (nedeterminističko polinomijalno vrijeme). Ili drugim riječima, problem je NP-težak ako je težak barem kao NP-kompletan problem.



Slika 2.4: Razredi složenosti

2.6. Optimizacijski algoritmi

Optimizacijske algoritme prema načinu implementacije možemo podijeliti u nekoliko skupina. [3]

Rekurzivni i iterativni algoritmi. Kod rekurzivnih algoritama rade se uzastopni rekurzivni pozivi najčešće se namjerom reduciranja složenosti instance problema (podijeli pa vladaj). Iterativni algoritmi obavljaju određeni broj iteraciji gdje se u svakoj iteraciji obavlja izračun za specifičan dio algoritma. Premda se iterativni algoritmi mogu implementirati rekurzivno i obrnuto, s obzirom na problem koji se rješava ponekad preferiramo samo jedan od ta dva oblika.

Slijedni i paralelni algoritmi. Ova podjela isključivo je vezana uz arhitekturu implementacije algoritma. Kod slijednih algoritama koraci algoritama obavljaju se slijedno, dok kod paralelnih algoritama problem se dijeli na potprobleme koji se onda paralelno izvode na višeprocesorskim računalima.

Deterministički i stohastički algoritmi. Ova podjela odnosi se na način donošenja odluka tijekom rada algoritma. Deterministički algoritmi imaju točno definiranu odluku s obzirom na trenutno stanje dok se kod stohastičkih algoritama ta odluka donosi nasumično. Većina heurističkim algoritama spada upravo u stohastične algoritme. U ovom radu bit ćemo fokusirani na stohastičnu optimizaciju.

Egzaktni i aproksimacijski algoritmi. Egzaktni algoritmi traže točno rješenje problema, odnosno globalni optimum funkcije cilja. Aproksimacijski algoritmi pokušavaju se samo približiti točnom rješenju što je više moguće, ili jednostavno rečeno, aproksimiraju rješenje. S obzirom na težinu prethodno opisanih problema, očito je da je skup problema rješivih egzaktnim pristupom poprilično mali zato što u većini slučajeva točno (egzaktno) rješenje nije moguće pronaći.

2.7. Stohastički optimizacijski algoritmi

Kao što je već prije spomenuto, stohastični optimizacijski algoritmi ne donose egzaktne odluke već nasumične. Najčešće dolaze u obliku raznih heuristika, a najčešće metaheuristika.

Metaheuristike ili moderne heuristike nastale su iz ideje poopćenja heuristika. Sve se heuristike temelja na logičnim pretpostavkama koji bi nam trebale biti vodilje do našeg rješenja. Najčešće se za svaki problem radi posebna heuristika jer svaki problem ima neku specifičnost koja je često ključna za optimalno rješenje stoga izrada heuristike nije uvijek lak posao. Takve heuristike jako rijetko pronalaze svoju primjenu osim za problem za koji su izgrađene. Kako bi se smanjio trud koji je potrebno uložiti za izradu heuristike, nastale su metaheuristike. Za razliku od heuristika, metaheuristike se zbog svoje apstraktnosti i općenitosti uz manju prilagodbu mogu primijeniti na širok spektar problema. Njihov rad temeljni se na jednom ili oba principa:

- postupna izgradnja rješenja iz pojedinih komponenti rješenja ili
- modifikacija postojećeg potpunog rješenja.

Metaheuristike se najčešće koriste kod optimizacijskih algoritama i to kao stohastični algoritmi. Posebno su prikladni kada razumijevanje problema koji se rješava nije dovoljno za izradu izravne heuristike. Nove metaheuristike stalno nastaju, pogotovo one nastale inspiracijom iz prirode ili stvarnog života. U nastavku slijedi nekoliko najčešće korištenih metaheuristika. [3]

1. **Lokalna pretraga** jedna ja od najjednostavnijih metaheuristika. Svoj princip rada temelji na restrikciji prostora pretrage. Pretraga počinje od početne točke.

Ako se unutar ograničenog prostora pronađe bolje rješenje od početnog, postupak se ponavlja u ograničenom okruženju novog rješenja, u suprotnom pretraga se zaustavlja.

2. **GRASP** koristi lokalnu pretragu koja je više puta pokrenuta iz početne točke koja je izgrađena slučajnim pohlepnim algoritmom. Ili drugim riječima, svaki put započinje pretragu iz drugog početnog rješenja.

3. **Tabu pretraga** prilikom pretraživanja nedavno posjećena rješenja pohranjuju se u $tabulistu$. Pretraga se kreće u smjeru najboljeg riješanja, ali pritom izbjegava rješenja pohranjena u tabu listu. Na takav način izbjegavaju se lokalni minimum ili ponavljanje ciklusa. Tabu lista je veličine k, gdje je k metaheuristički parametar.

4. **Simulirano kaljenje** je metaheuristika koja je inspirirana kaljenjem metala, postupku kojem kristali u prirodi teže u stanje minimalne energije. Algoritam obilazi prostor pretrage lokalnom pretragom, ali uz kontrolu pomoću parametra $temperature$. Što je temperatura viša to je veća vjerojatnost da će pretraga krenuti u smjeru lošijeg rješenja. U početku temperatura je visoka te se tako sprječava zaustavljanje u lokalnom optimumu, dok je u završnoj fazi temperatura niska, tj. smjer pretrage je smjeru najboljeg rješenja.

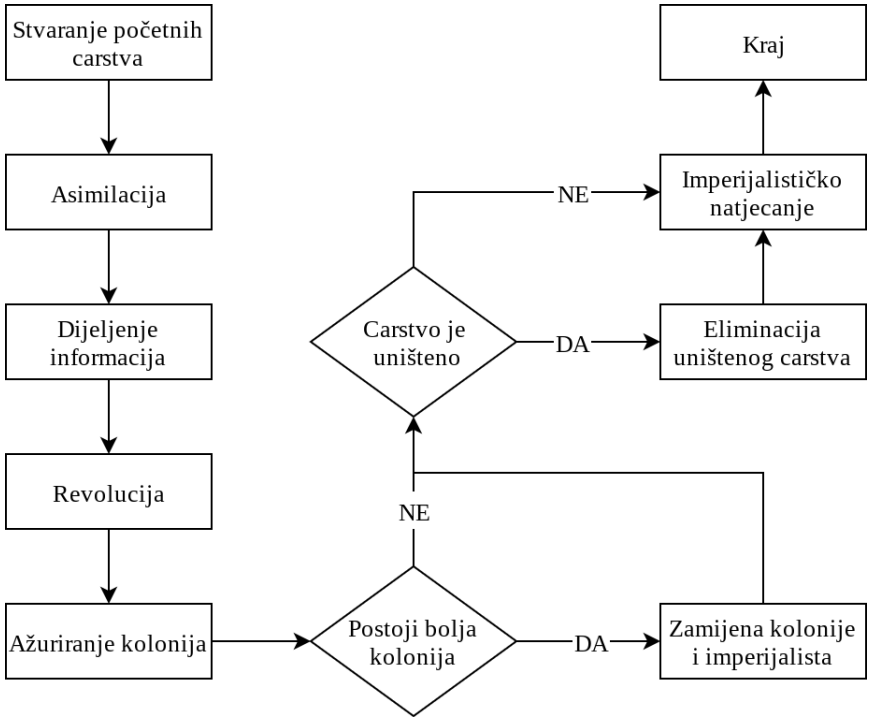
5. **Genetski algoritam** je metaheuristika inspirirana teorijom evolucije. U prirodi nužna je konstanta prilagodba na način da svaka sljedeća generacija sadrži najbolja svojstva prethodne generacije. Temelji se na operatorima križanja, nasljeđivanja, mutaciji te selekciji.

6. **Algoritam kolonije mrava** je metaheuristika inspirirana životom i organizacijom mravlje kolonije. Koordiniranim radom veliki broj mrava može obaviti kompleksne operacije. Mehanizam koji ovaj algoritam bazira koristi mirisne tvari - $feromone$. Mravi koriste feromone za pronalazak najkraćeg puta od mravinjaka do mjesta gdje se nalazi hrana.

3. Imperijalistički optimizacijski algoritam

Imperijalistički optimizacijski algoritam je metaheuristički optimizacijski algoritam inspiriran velikim carstvima i njihovim međusobnim bitkama i osvajanjima. Algoritam jedinke naziva državama koje se dijele na imperijalističke te kolonije. Svaka generacija zapravo predstavlja desetljeće proteklog vremena, a kroz jedno desetljeće (generaciju) carstva se poboljšavaju te međusobno bore za novu koloniju. [2]

Struktura rješenja u ovom primjeru problema definirana je permutacijskim genotipom, odnosno poljem cijelih brojeva u kojem nema ponavljanja dva ista broja. Duljina polja definirana je problemom, npr. ako rješavamo problem trgovačkog putnika s 30 gradova, duljina polja biti će 30. **Slika 3.1** prikazuje logički slijed izvođenja algoritma.



Slika 3.1: Imperialistički algoritam

3.1. Stvaranje početnih carstva

Na početku svog rada algoritam stvara početna carstva. Svaka jedinka populacije predstavlja jednu državu koja predstavlja jedno N-dimenzijsko optimizacijsko rješenje gdje je p_i varijabla koja se optimizira.

$$drzava = [p_1, p_2, ..., p_N] \tag{3.1}$$

Redoslijed brojeva unutar polja predstavlja redoslijed izvođenja poslova.
Koristeći funkciju cilja evaluiramo svaku jedinku.

$$c_i = f(drzava) = f(p_1, p_2, ..., p_N) \tag{3.2}$$

Parametrom N_{imp} je definirano koliko jedinki populacije postaju imperijalističkim državama dok preostale postaju kolonijama, tj. države podređene imperijalističkima. Uzimamo prvih N_{imp} najboljih država po kriteriju vrijednosti funkcije cilja te ih normaliziramo.

$$C_n = max_i c_i - c_n \tag{3.3}$$

Posljedica normalizacije je da jedinke koje su bile lošije imaju manju vrijednost normalizirane funkcije cilja. To nam pomaže pri određivanju moći imperijalističke države. Što je bolja funkcija cilja to je država moćnija. Moć imperijalističke države računamo prema izrazu

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right|. \tag{3.4}$$

Primijetimo da je zboj moći svih imperijalističkih država jednaka 1. Sada jednostavno možemo izračunati koliko kolonija pripada svakom carstvu.

$$NC_n = round\{p_n * N_{col}\} \tag{3.5}$$

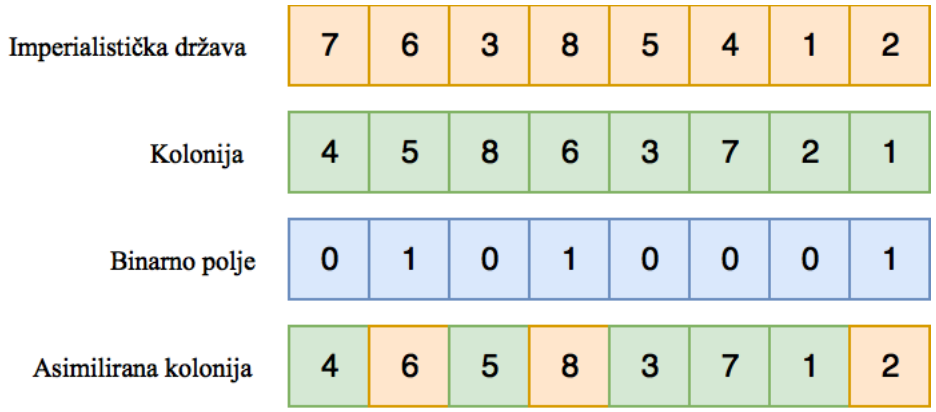
Sada nasumično odabiremo NC_n kolonija koje će pripasti imperijalističkoj državi.

3.2. Asimilacija

Kako bi se svaka kolonija poboljšala, ona se jednim dijelom pokušava prisličiti svojoj imperijalističkoj državi. To se radi na način da se prvo gradi binarno polje duljine N s gustoćom jedinica β. Gustoća jedinica se mijenja kroz desetljeća (generacije) od vrijednosti β_{min} do β_{max} prema izrazu:

$$\beta = \beta_{max} - \frac{(MaxGeneracija - Generacija) * (\beta_{max} - \beta_{min})}{MaxGeneracija - 1}. \tag{3.6}$$

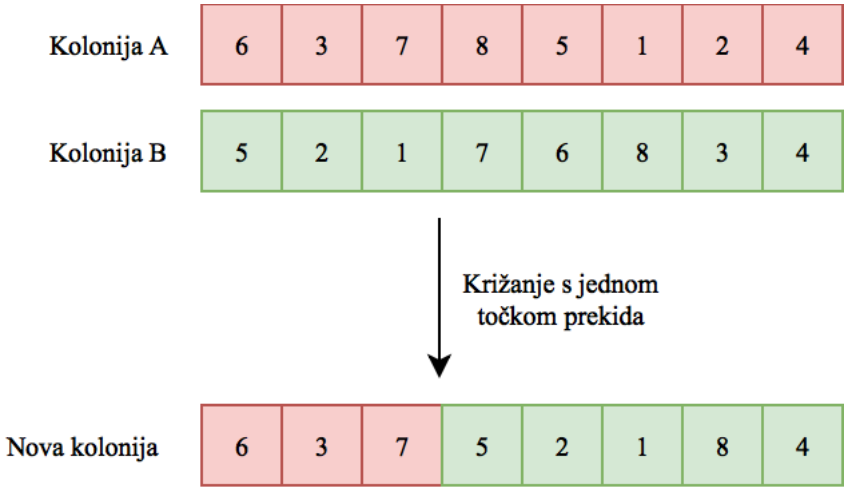
Kada je struktura rješenja permutacijski genotip, na svakom indeksu binarnog polja gdje se pojavljuje jedinica uzima se vrijednost imperijalističke države na istom indeksu te se preslikava u polje kolonije (**slika 3.2**). U slučaju da u koloniji na nekom drugom indeksu ta vrijednost već postoji, ona se zamjenjuje za mjesta s prethodnom vrijednosti kolonije na mjestu indeksa asimilacije. Npr. recimo da imamo broj 1 na indeksu 1, te prilikom asimilacije dolazi do ponavljanja broja 1 na indeksu 5. Tada ćemo vrijednost s indeksa 5 zamijeniti s jedinicom na indeksu 1. Tako se i dalje vrijednosti ne ponavljaju više od jednom.



Slika 3.2: Asimilacija

3.3. Dijeljenje informacija

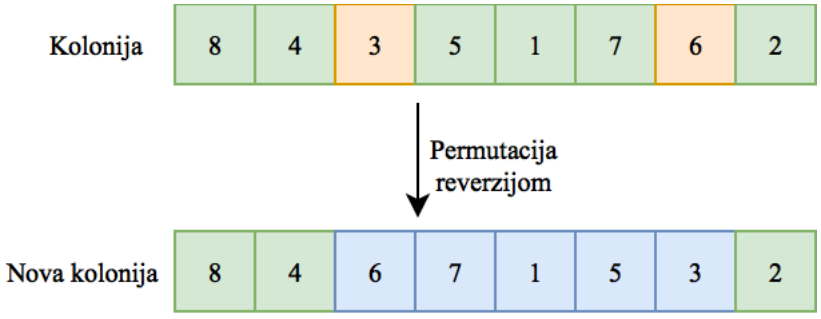
Još jedan od načina poboljšanja jedinki je dijeljenje informacija. Parametrom *pCrossover* određen je udio populacija koji sudjeluje u tom procesu. U svakoj iteraciji biramo dvije jedinke, a informacije će dijeliti samo ona koja je bolja (selekcija binarnim dvobojem). Samo dijeljenje informacija ostvareno je korištenjem operatora križanja. Korišteni su svi dostupni operatori križanja programskog sustava za evolucijsko računanje (ECF [1]) **Slika 3.3** prikazuje primjer dijeljenja informacija pomoću operatora križanja s jednom točkom prijeloma. Isto kao i kod asimilacije, i ovdje pazimo da se niti jedna vrijednost ne ponovi više nego jednom. Redom zamjenjujemo mjesta svim vrijednostima kolonije B koja će biti duplicirana nakon križanja s kolonijom A u točki prijeloma pri nastanku nove kolonije. Npr. ako će se broj 1 dvaput ponoviti nakon križanja, zamijenit ćemo broj 1 kolonije B s vrijednosti koja se nalazi na indeksu gdje će se nalaziti broj 1 kolonije A nakon križanja.



Slika 3.3: Primjer dijeljenja informacija

3.4. Revolucija

U svakoj iteraciji algoritma bira se nekoliko kolonija nad kojima se u svrhu mogućeg poboljšanja jedinke permutiraju vrijednost polja. Udio populacije nad kojima se provodi revolucija određen je parametrom *pRevolution*. **Slika 3.4** prikazuje primjer revolucije korištenjem reverznog operatora mutacije.



Slika 3.4: Primjer revolucije

3.5. Ažuriranje kolonija

Nakon što smo nad svakim carstvom proveli asimilaciju, te nad dijelom populacije dijeljenje informacija te revolucija ažuriramo postojeće kolonije. Biramo one novonastale jedinke koju su bolje od postojećih kolonija te ih zamjenjujemo pazeći pritom da broj kolonija pojedinog carstva ostane nepromijenjen.

3.6. Zamjena kolonije i imperijalista

Nakon svih provedenih promjena nad carstvima može doći do situacije kada imperijalistička država više nije najbolja jedinka carstva već je to neka od kolonija. Tada imperijalistička država postaje kolonijom, a kolonija dolazi na vodeću poziciju carstva.

3.7. Eliminacija uništenih carstva

Tijekom iteracija može doći do situacije da neko carstvo nema niti jednu koloniju zbog posljedica prijašnje iteracije ili jednostavno zbog nedovoljno moći prilikom inicijalizacije početnih carstva. Tada se to carstvo smatra uništenim te imperijalistička država postaje kolonijom nekog nasumično odabranog carstva.

3.8. Imperijalističko natjecanje

Svaku iteraciju carstva pokušavaju steći novu koloniju kako bi proširili svoje carstvo. To se odvija tako što se bira najslabija kolonija najslabijeg carstva na osnovu kriterija ukupne snage. Ukupna snaga carstva određena je zbrojem funkcije cilja imperijalističke države s umnoškom parametra ζ i srednje vrijednosti iznosa funkcija cilja kolonija tog carstva. Parametar ζ pritom određuje koliko će iznos funkcija cilja kolonija utjecati na ukupnu snagu carstva.

$$N = \text{broj kolonija carstva}$$

$$TC_n = f(imperijalist_n) + \zeta * \frac{\sum_{i=1}^{N} f(kolonija_i)}{N} \tag{3.7}$$

Nakon što se pronađe najslabija kolonija najslabijeg carstva, pobjedničko carstvo odabrano je korištenjem proporcionalne selekcije (*Roullette Wheel Method*). Ideja iza ove selekcije je da sve jedinke postavimo na kolo te da pritom bolja jedinka zauzima veći dio površine kole. Potom "zavrtimo" kolo te promatramo na kojoj jedinki će se kolo zaustaviti.

3.9. Kraj rada algoritma

Na kraju iteracije ako postoji jedinka koja zadovoljava postavljeni kriterij zaustavljanja, tj. ima odgovarajuću vrijednost funkcije cilja, algoritam uspješno završava svoj rad.

3.10. Polje decimalnih brojeva

Algoritam kao strukturu rješenja problema može koristiti i polje decimalnih brojeva. U tom slučaju problem koji se rješava više nije optimizacija redoslijeda. Asimilacija ne mijenja svoju funkcionalnost već način rada zato što mijenja vrijednosti polja kako bi koloniju prisličila imperijalističkoj državi. Ostatak algoritma funkcionira na jednak način kao prethodno opisano.

4. Rezultati

Obrada rezultata algoritma bit će provedena u nekoliko koraka. Prvo određujemo za koji broj evaluacija prosječna najbolja rješenja algoritma konvergiraju. Zatim ćemo ispitati kako različite vrijednosti parametara utječu na pronalazak rješenja kroz 20 pokretanja algoritma. Sva rješenja dobivena su rješavanjem problema trgovačkog putnika koji pokušava obići 130 gradova ($ch130 - Churritz$). Najbolje poznato rješenje tog problema iznosi 6110 [4].

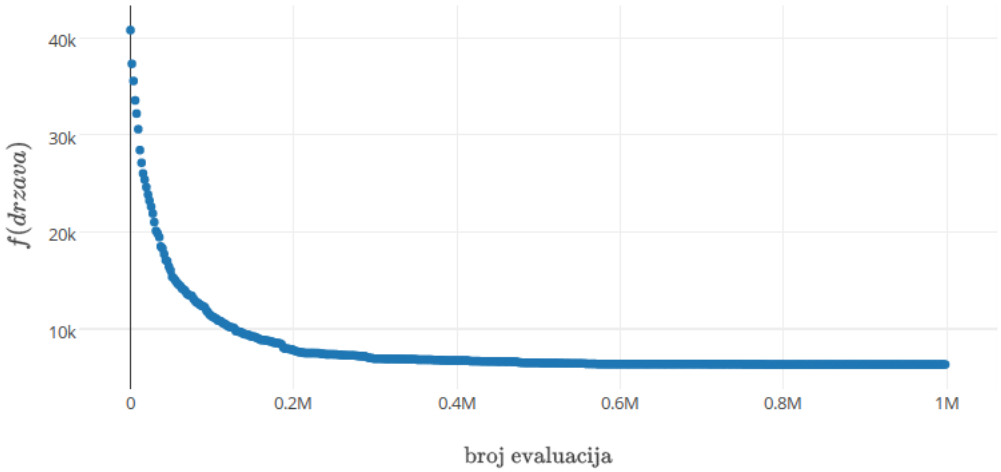
4.1. Kriterij zaustavljanja

Kriterij zaustavljanja određen je brojem evaluacija za koji rješenje optimizacije konvergira. Kako bismo odredili taj kriterij rad algoritma ograničen je velikim brojem evaluacija (1000000) te inicijaliziran sljedećim početnim parametrima (**Tablica 4.1**):

Naziv parametra	Iznos
β_{min}	0.25
β_{max}	0.75
ζ	0.1
N_{imp}	13
N_{pop}	130
$maxGeneracija$	4000
$pCrossover$	1
$pRevolution$	1
vjerojatnost mutacije	0.3

Tablica 4.1: Početni parametri

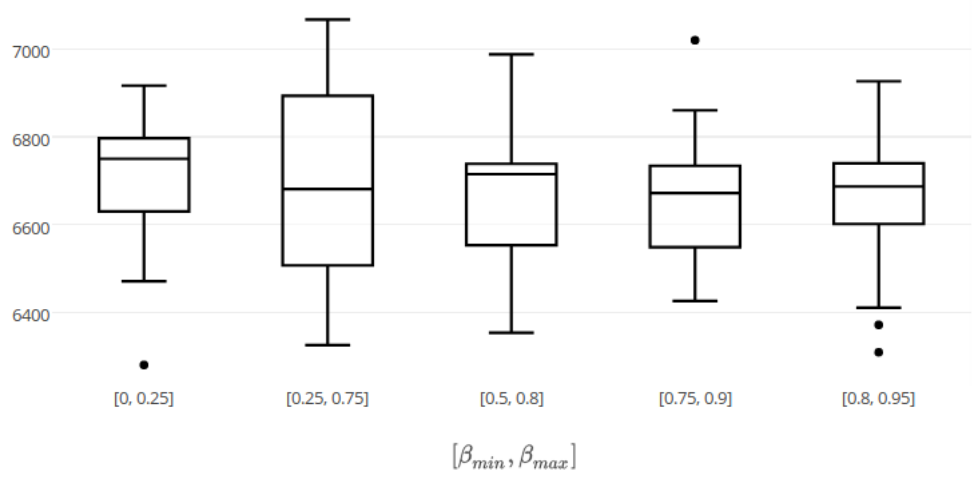
Slika 4.1 prikazuje kako se prosječna najbolja rješenja mijenjaju s brojem evaluacija. Primjećujemo kako za početne parametre metaheuristike algoritam pronalazi optimum nakon 600000 evaluacija. Stoga ćemo za testiranje ostalih parametara pretpostaviti da je ograničenje na 600000 evaluacija sasvim dovoljno.



Slika 4.1: Konvergencija s brojem evaluacija

4.2. Ovisnost o β

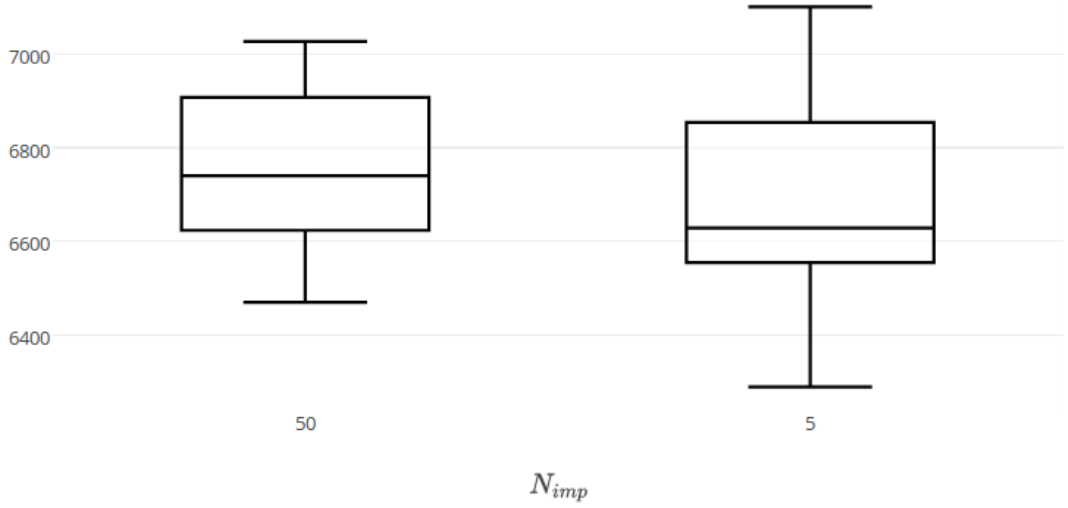
β parametar izračunat je u procesu asimilacije (**3.2**). Njegova vrijednost nalazi se na intervalu [β_{min}, β_{max}], a točan iznos definiran je brojem trenutne generacije u odnosu na broj generacija kojom je algoritam ograničen $MaxGeneracija$ (**izraz 3.6**). Određuje u kojem postotku će se kolonija prisličiti imperijalističkoj državi. S obzirom na rezultate (**Slika 4.2**) vidimo da nije jednostavno odrediti najbolji interval tog parametra. Za interval [0, 0.25] dobivamo relativno lošije rezultate jer ne slijedimo heuristiku u toliko velikoj mjeri te češće završavamo u lokalnim optimumima. Kada je taj interval bliže 100% ne dobivamo puno bolje rezultate iz istog razloga, samo što je uzrok pretjerano praćenje heuristike. Najpogodniji interval nalazi se između navedene dvije krajnosti. Koliko je dobar rezultat ovisi da li algoritam, u trenutku kada počinje više slijediti heuristiku, krenuo prema lokalnom optimumu ili globalnom optimumu (ili nekom još boljem lokalnom optimumu).



Slika 4.2: Ovisnost rješanja o β parametrima

4.3. Ovisnost o N_{imp}

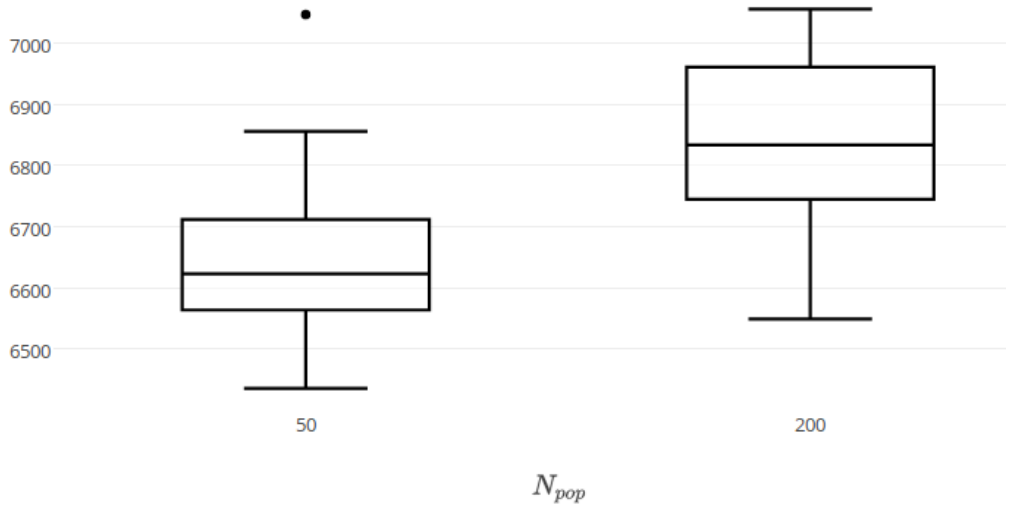
Parametrom N_{imp} određujemo koliko imperijalističkih država želimo. Primjećujemo da ako veličina populacije ostaje jednaka, a broj imperijalističkih država se poveća, rješenja budu lošija. Razlog tome je to što smanjujemo broj kolonija svake imperijalističke države, a time smanjujemo i prostor pretrage svakog carstva. Posljedica toga je to što carstva pronalaze najčešće samo okolne lokalne optimume.



Slika 4.3: Promjena rezultat u ovisnosti o N_{imp} parametru

4.4. Ovisnost o N_{pop}

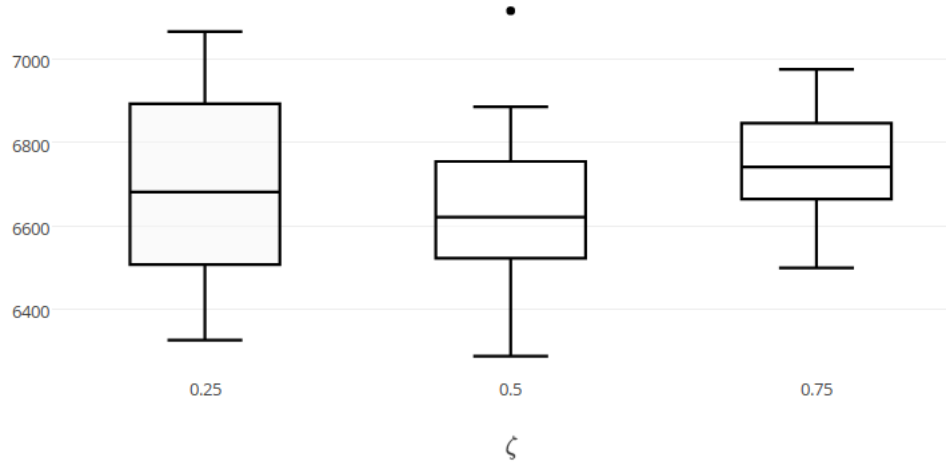
Parametrom N_{pop} određujemo veličinu populacije. Za manju veličinu populacije dobivamo bolja rješenja, ali to je zato što se veličina populacije mijenja paralelno s veličinom problema koji rješavamo.



Slika 4.4: Ovisnost rješenja o N_{pop} parametru

4.5. Ovisnost o ζ

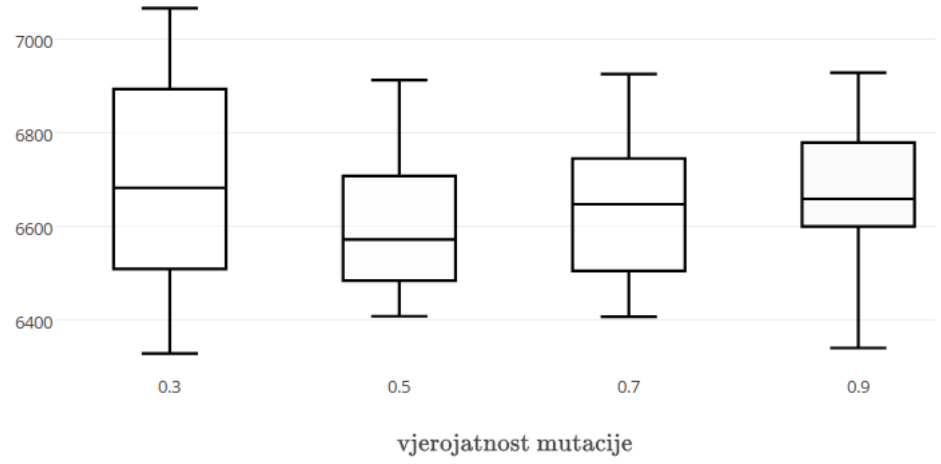
Parametrom ζ određujemo koliko moć svake kolonije carstva utječe na ukupnu moć imperijalističke države (**izraz 3.7**). Na **slici 4.5** prikaza je ovisnost pronađenog rješenja i parametra ζ. U procesu imperijalističkog natjecanja (**3.8**) najslabije carstvo određuje se na osnovu ukupne moći carstva, te se onda natjecanje vrši nad najslabijom kolonijom najslabijeg carstva. Ako je parametar ζ prevelik dolazi do situacije da neka slaba carstva zapravo ne budu među najslabijima. Loše kolonije tih carstva se u sljedećoj iteraciji ne uspijevaju asimilirati s boljim imperijalističkim državama te i dalje kreću k lošem rješenju.



Slika 4.5: Ovisnost rješenja o ζ parametru

4.6. Ovisnost o vjerojatnosti mutacije

Slika 4.6 uspoređuje rezultate za različite vrijednosti vjerojatnosti mutacije. Kada je vjerojatnost mutacije 0.5 u prosjeku rješenja su bolja. Za sve veće vrijednosti vjerojatnosti mutacije nedostaje ono "fino podešavanje" koje je potrebno u završnom dijelu rada algoritma.

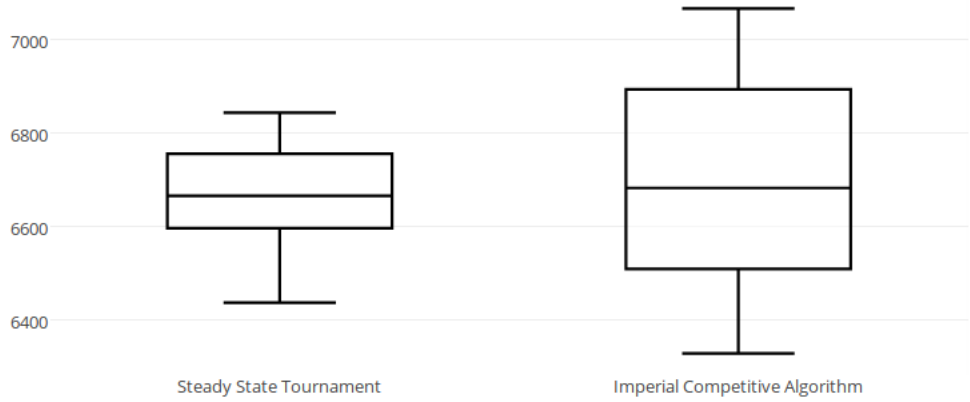


Slika 4.6: Ovisnost rješenja o vjerojatnosti mutacije

4.7. Usporedba s drugim algoritmima ECF-a

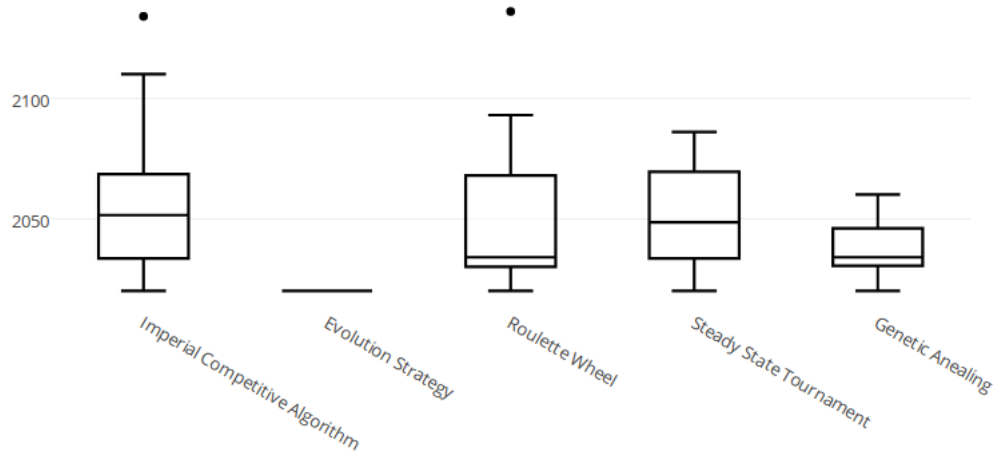
U sklopu programskog sustava za evolucijsko računanje (ECF [1]) jedini algoritam koji je uspio dobiti približno dobra rješenja na danom problemu kao i imperijalistički

algoritam je *Steady State Tournament* algoritam. Imperijalistički algoritam uspio je pronaći bolja rješenja premda nije toliko precizan u većini slučajeva kao *Steady State Tournament* algoritam.



Slika 4.7: Usporedba algoritama

Na manje zahtjevnom problemu kada trgovački putnik pokušava obići 29 gradova ($bays29$ - 29 gradova u Bavariji [4]) svi algoritmi ECF-a [1] uspjeli su pronaći prihvatljiva rješenja (**slika 4.8**). U ovom slučaju imperijalistički algoritam uspio je pronaći optimalno rješenje (2020 [4]), ali u prosjeku većina rješenja su mu lošija naspram drugih algoritama. Približna rješenja pronašao je i $SteadyStateTournament$. $GeneticAnealing$ te $RouletteWheel$ su s obzirom na prosjek rješenja jednako dobri dok je $EvolutionStrategy$ uspio svaki put pronaći optimalno rješenje i to u najkraćem vremenu.



Slika 4.8: Usporedba algoritama

5. Zaključak

Problem redoslijeda, ovisno o broju elemenata, predstavlja jako kompleksan problem. Pogotovo kada savršeno rješenje ne postoji potraga za onim sljedećim najboljim težak je put s puno prepreka, odnosno lokalnih optimumima. Na našu sreća, rješenja su svuda oko nas. Inspirirani svakodnevnim pojavama oko nas možemo osmisliti metaheuristike koje nas vode k točnom rješenju. Imperijalistički optimizacijski algoritam savršeni je primjer takve metaheuristike inspirirane imperijalističkim borbama i osvajanjima. Pomoću parametra $beta$ u početnoj fazi rada algoritam izbjegava zamke lokalnih optimuma, a operatorima križanja i mutacije istražuje prostor pretrage vođen prema dobrom rješenju pomoću postupka asimilacije. Zbog svoje metaheurističnosti algoritam je primjenjiv na širi spektar problema, a s obzirom na rezultate, algoritam uspijeva pronaći zadovoljavajuća rješenja. Premda izazov predstavlja prilagoditi sve parametre problemu koji rješavamo, imperijalistički algoritam je više nego dobra konkurencija ostalim optimizacijskim algoritmima.

LITERATURA

[1] izv. prof. dr. sc. Domagoj Jakobović. *ECF - Evolutionary Computation Framework*. URL http://ecf.zemris.fer.hr/html/index.html.

[2] M. Mazinani R. Shafaei M. Rabiee, Reza Sadeghi Rad. *An intelligent hybrid meta-heuristic for solving a case of no-wait two-stage flexible flow shop scheduling problem with unrelated parallel machines*. URL http://www.zemris.fer.hr/~yeti/studenti/izvori/An%20intelligent%20hybrid%20meta-heuristic%20for%20solving%20a%20case%20of%20no-wait%20two-stage%20flexible%20flow%20shop%20scheduling.

[3] Goran Molnar. *Heuristički i metaheuristički postupci optimizacije*. URL https://www.fer.unizg.hr/_download/repository/Goran_Molnar_KDI.pdf.

[4] Heidelberg University. *Discrete and Combinatorial Optimization*. URL https://www.iwr.uni-heidelberg.de/groups/comopt/.

Ostvarenje imperijalističkog optimizacijskog algoritma u programskom sustavu za evolucijsko računanje

Sažetak

Rad opisuje problem optimizacije te pristup njegovom rješavanju. Objašnjava kako popularne metahueristike generaliziraju problem jednostavnim modelom te dublje opisuje metaheuristiku inspiriranu imperijalističkim borbama carstava. Sadrži pregled rezultata optimizacije imperijalističkog algoritma s obzirom na parametre metaheuristike. Kraj rada sadrži zaključak kroz kratki osvrt na problem optimizacije te dublje opisane imperijalističke metaheuristike.

Ključne riječi: optimizacija, stohastičan, algoritam, imperijalstički, metaheuristika

Imperialistic Competitive Algorithm

Abstract

This final work contains explanation of optimisation problems and how to solve them. It explains how popular metaheuristics generalize the problem of optimisation trough simple problem model and offers deeper explanation of metaheuristics inspired by imperialistic competition of empires. There is also a summary of imperialistic algorithm results and how does the metaheurustic parameters affect them. At the end there is a short summary of optimisation problems and final conclusion about imperialistic competitive algorithm.

Keywords: algorithm, imperialistic, competitive, stochastic, metaheuristic